REST Driven Data Processing in the Atmospheric Composition Processing System

Marty Brandon[1] <mbrandon@sesda2.com>, Brian Duggan [1] <brian.duggan@nasa.gov>, Phil Durbin[1] <phillip.durbin@nasa.gov>, Andrew Laughland[1] <alaughland@sesda2.com>,
Arnold Martin[3] <amartin@innovim.com>, Val Soika[1] <vsoika@sesda2.com>, Curt Tilmes[2] <curt.tilmes@nasa.gov>, Michael Walters[1] <michael.a.walters@nasa.gov>
1:ADNET Systems, Inc. 2:NASA/GSFC, 3: INNOVIM, Inc., 4: Wyle, Inc

A play in two acts

Prologue

The Hypertext Transfer Protocol (HTTP) provides a rich vocabulary for manipulating and sharing resources between computing nodes. Representational State Transfer (REST) is an architectural style that uses HTTP as its basis. The Atmospheric Composition Processing System uses a REST architecture in order to process and generate data collected from the Ozone Monitoring Instrument on the Aura spacecraft and the Ozone Mapping and Profiler Suite (OMPS) on the NPP spacecraft. This choice of architecture allows the system to use many existing tools and concepts for data processing that are already employed as part of the World Wide Web, and to be forward-compatible with evolving HTTP-based standards for provenance representation, granule searching, and data distribution.

Cast of Characters

Restmd : Stores metadata for files and other permanent artifacts.
Restapp : Stores metadata for applications, complementing a version control system.
Sched : Stores ephemeral artifacts representing the current plans, tasks, jobs.
SimpleAuth : Performs authentication and authorization for other servers.
Minions : Run the jobs, return the results.
Yars : The archive, stores all data.

Act I

in which a granule is produced

enter ***Scientist*** *and* ***Sched***	
Scientist : Sched, compute the Total Ozone for OMI orbit 99!	POST /plan
Sched : As you wish.	200 OK
Scientist : Is it running?	GET /plan/1234
Sched : Not yet.	["State" : "Planned"]
Scientist : What else is running?	GET /plan/search
Sched : a lot of things	[..list of plans..]
enter 50 ***Minions***	
Minion 23 : Sched, what can I do to help?	GET /job
Sched : Here, run this job.	["OMTO3" : "1.1", "Orbit" : "99"]
Minion 23 *runs the job, and it finishes.*	
enter ***Restmd*** *and* ***Yars***	
Minion 23 : Restmd, take these artifacts.	POST /appevent
Restmd : Thanks.	200 OK
Minion 23 : Yars, take this data.	PUT /file
Yars : Thanks.	200 OK
Minion 23 : Sched, I'm done.	POST /job/status
Sched : Great.	200 OK
Scientist : Is it done?	GET /plan/1234
Sched : Yes.	["State" : "Complete"]
Scientist : Then, where is the data?	
curtain falls	

Act II

in which a granule is downloaded

enter ***Scientist*** *and* ***Restmd***	
Scientist : Where is OMTO3 orbit 99?	POST /file/search
Restmd : It is in the archive at this URL.	["Location" : "/abc"]
Scientist : And in what environment was it generated?	POST /appevent/search
Restmd : Ask the archive	["Location" : "/abcevent"]
Scientist : Archive?	GET /status
enter ***Yars***	
Yars : I'm here.	200 OK
Scientist : Give me the data.	GET /abc
Yars : Here it is.	[data]
Scientist : and the execution environment.	GET /abcevent
Yars : Here it is.	[more data]
curtain falls	

Behind the Scenes

Servers and client use a common framework which provides consistent techniques for logging, API instrospection, authentication and authorization, status reporting, and dependency resolution. A web user interface makes use of the clients, and provides a proxy directly to the backend, so that external clients can connect to the services.

Restmd : File/granule metadata.

```
$ restmdclient api
---
- GET /appevent
- GET /archiveset
- GET /esdt
- GET /file/(*key)
- GET /file/identify/(.filename)
- GET /file/secondary/:id/:tag
- GET /file_metadata/(*key)
- GET /files/count/archiveset/:min_ingest/:max_ingest
- POST /esdt
- POST /file_metadata
...
```

- Restmd provides RSS and Atom feeds compatible with Discovery standards.
- **Production Rules** use Restmd to find inputs when generating granules.

Restapp : App metadata.

```
$ restappclient api
---
- GET /app
- GET /app/(*key)
- GET /app/(.name)/(.version)
- GET /app_dynamic_file/(*key)
- GET /app_person/(*key)
- GET /app_runtime_parameters/search
- GET /persons/search
- POST /app/(*key)
- POST /app_static_input_files/search
- POST /persons/search
...
```

- Used in conjunction with a version control system for delivery of apps.

Sched : Schedules apps to run.

```
$ schedclient api
---
- GET /job
- GET /plan
- GET /task
- POST /cancel_jobs
- POST /expire_stalled_jobs
- POST /granularities/search
- POST /rerun_tasks
- POST /retry_tasks
- POST /state_change
- POST /tasks/search
...
```

- Tracks the states of jobs, plans and tasks to be run on the minions.

MinionRelay : Real time minion monitoring.

```
$ minionrelay api
---
- GET /status
- GET /api
- GET /version
- GET /min
- GET /min/:which
- GET /mins
- GET /tellmin/:which/(.*what)
- GET /log/:lines
```

- Maintains websocket connections, allowing Sched to monitor progress and start/stop jobs instantly.

Dataflow : Visualize and query the graph of dependencies.

```
$ acpsdataflowclient api
---
- GET /:noun
- GET /:noun/:instance
- GET /:noun/:instance/:action
- GET /app/:app/cm
- GET /app/:app/info
- GET /archive_set/:archive_set/graph
- GET /archive_set/:archive_set/graph/:output
- GET /archive_set/:archive_set/graph/printable/print
- GET /archive_set/:instance/rules
- GET /esdt/:esdt/info
- GET /esdt/:esdt/stats
```

- Deduces all ascendants and descendants of a given product, at the dataset level.

Minions : Processing nodes

- The minions use the same server APIs as an external client.
- Minions initiate HTTP transactions, but do not listen for any.
- There is no centralized minion management, just a registry in Sched based on incoming connections.
- External hosts can become minions, as can virtual hosts (e.g. in a cloud).
- Minions get jobs from Sched and applications from Restapp. They store granules and artifacts in Yars, and metadata in Restmd.
- Minions are responsible for sending details of the execution environment, inputs, outputs, and other provenance-related artifacts.
- Minions may also initiate websocket connections with MinionRelay to allow jobs to be monitored.
- Minions use **Data::Downloader** to maintain a fixed-size LRU cache of applications and data.
- **Data::Downloader** can subscribe to RSS feeds and use Discovery Service APIs to retrieve data from other sources.

Yars : Archive of granules and artifacts.

```
$ yarsclient api
---
- PUT /file/(.filename)/:md5
- GET /bucket_map
- GET /disk/usage
- GET /file/(.filename)/:md5
- GET /file/:md5/(.filename)
- GET /log/:lines
- GET /servers/status
- POST /check/manifest
- POST /disk/status
- DELETE /file/(.filename)/:md5
- DELETE /file/:md5/(.filename)
```

- Uses a **distributed hash table** for storage. Because there is no central server, new hosts and disks may be added just by updating the hash table.

SimpleAuth : Check authentication and authorization

```
$ simpleauth api
---
- GET /status
- GET /api
- GET /version
- GET /auth
- GET /authz/resources/(.user)/(.action)/(*resourceregex)
- GET /authz/user/(.user)/(.action)/(*resource)
- GET /log/:lines
```

- Interfaces with EOS Authentication LDAP service.

Plot Overview

- The ACPS is a **distributed** architecture. There is no central control; servers all act independently.
- Communication is all done using REST calls. The message payload for the calls is encoded in JSON, but other encodings may be requested (e.g. XML, YAML) via HTTP Accept Headers.
- MD5 digests for data are used during storage, transfer, and downloading, to guarantee data integrity.
- The internal interfaces between the servers are the same as the external APIs.
- The environment is captured during execution of applications. This provenance data can then be used later to reproduce results.